Package 'myClim'

May 8, 2023

Type Package

Title Microclimatic Data Processing

Version 1.0.1

URL http://labgis.ibot.cas.cz/myclim/index.html,
https://github.com/ibot-geoecology/myClim

Description Handling the microclimatic data in R. The 'myClim' workflow begins
at the reading data primary from microclimatic dataloggers,
but can be also reading of meteorological station data from files.
Cleaning time step, time zone settings and metadata collecting is the next step of the work flow.
With 'myClim' tools one can crop, join, downscale, and convert microclimatic data formats, sort them into localities,
request descriptive characteristics and compute microclimatic variables.
Handy plotting functions are provided with smart defaults.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Depends R (>= 3.5)

Imports stringr, lubridate, tibble, dplyr, purrr, tidyr, ggplot2,
ggforce, viridis, runner, plotly, zoo, methods

Additional_repositories https://ibot-geoecology.github.io/drat

Suggests rmarkdown, knitr, kableExtra, rTubeDB, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Matěj Man [aut],
Vojtěch Kalčík [aut, cre],
Martin Macek [aut],
Jan Wild [aut],
Martin Kopecký [aut],

Josef Brůna [aut],
Lucia Hederová [aut],
Institute of Botany of the Czech Academy of Sciences [cph]

Maintainer Vojtěch Kalčík <Vojtech.Kalcik@ibot.cas.cz>

Repository CRAN

Date/Publication 2023-05-08 17:40:02 UTC

R topics documented:

`mc_agg` *Aggregate data by function*

Description

mc_agg has two basic uses:

- aggregate (upscale) time step of microclimatic records with specified function (e. g. 15 min records to daily mean);
- convert myClim object from Raw-format to Agg-format see myClim-package without time-series modification, this behavior appears when `fun=NULL`, `period=NULL`.

Usage

```
mc_agg(
  data,
  fun = NULL,
  period = NULL,
  use_utc = TRUE,
  percentiles = NULL,
  min_coverage = 1,
```

```
  custom_start = NULL,
  custom_end = NULL,
  custom_functions = NULL
)
```

Arguments

data	cleaned myClim object in Raw-format: output of mc_prep_clean() or Agg-format as it is allowed to aggregate data multiple times.
fun	aggregation function; one of ("min", "max", "mean", "percentile", "sum", "range", "count", "coverage") and functions defined in custom_functions. See details of custom_functions argument. Can be single function name, character vector of function names or named list of vector function names. Named list of functions allows apply different function(s) to different sensors e.g. list(TMS_T1=c("max", "min"), TMS_T2="mean", TMS_T3_GDD="sum") if NULL records are not aggregated, but myClim object is only converted to Agg-format without modifing time-series. See details.
period	Time period for aggregation - same as breaks in cut.POSIXt, e.g. ("hour", "day", "month"); if NULL then no aggregation There are special periods "all" and "custom". Period "all" returning single value for each sensor based on function applied across all records within the sensor. Period "custom" aggregates data in yearly cycle. You can aggregate e.g. water year, vegetation season etc. by providing start, end datetime. See custom_start and custom_end parameters. The output of special periods "all" and "custom"are not allowed to be aggregated again in mc_agg() function, regardless multiple aggregations are allowed in general. Start day of week is Monday.
use_utc	default TRUE using UTC time, if set FALSE, the time is shifted by offset if available in locality metadata. Shift can be e.g. to solar time mc_prep_solar_tz() or political time with custom offset mc_prep_meta_locality()). Non-UTC time can by used only for aggregation of the data with period shorter than day (seconds, minutes, hours) into period day and longer.
percentiles	vector of percentile numbers; numbers are from range 0-100; each specified percentile number generate new virtual sensor, see details
min_coverage	value from range 0-1 (default 1); the threshold specifying how many missing values can you accept within aggregation period. e.g. when aggregating from 15 min to monthly mean and set min_coverage=1 then a single NA value within the specific month cause monthly mean = NA. When min_coverage=0.9 then you will get your monthly mean in case there are no more than 10 % missing values, if there were more than 10% you will get NA. Ignored for functions count and coverage
custom_start	date of start, only use for custom period (default NULL); Character in format "mm-dd" or "mm-dd H:MM" recycled in yearly cycle for time-series longer than 1 year.
custom_end	date of end only use for custom period (default NULL); If NULL then calculates in year cycle ending on custom_start next year. (useful e.g. for hydrological

year) When custom_end is provided, then data out of range custom_start-custom_end are ignored. Character in format "mm-dd" or "mm-dd H:MM". custom_end row (the last record) is not included. I.e.complete daily data from year 2020 ends in 2021-01-01 custom_end="01-01".

custom_functions
: user define one or more functions in format list(function_name=function(values){...}); then you will feed function_name(s) you defined to the fun parameter. e.g. custom_functions = list(positive_count=function(x){length(x[x>0])}), fun="positive_count",

Details

Any output of mc_agg is in Agg-format. That means the hierarchical level of logger is removed (Locality<-Logger<-Sensor<-Record), and all microclimatic records within the sensors are on the level of locality (Locality<-Sensor<-Record). See myClim-package.

In case mc_agg() is used only for conversion from Raw-format to Agg-format (fun=NULL, period=NULL) then microclimatic records are not modified. Equal step in all sensors is required for conversion from Raw-format to Agg-format, otherwise period must be specified.

When fun and period are specified, microclimatic records are aggregated based on function into new period. Aggregated time step is named after the first time step of selected period i.e. day = c(2022-12-29 00:00, 2022-12-30 00:00...); week = c(2022-12-19 00:00, 2022-12-28 00:00...); month = c(2022-11-01 00:00, 2022-12-01 00:00...); year = c(2021-01-01 00:00, 2022-01-01 00:00...). When first or last period is incomplete in original data, the incomplete part is extended with NA values to match specified period. I.e. when you want to aggregate to monthly mean, but your time-series starts on January 15 ending December 20, myClim will extend the time.series to start January 1, ending December 31. Then you can decide what to do with the NAs, see parameter min_coverage.

Empty sensors with no records are excluded. mc_agg() return NA for empty vector except from fun=count which returns 0. When aggregation functions are provided as vector or list e.g. c(mean, min, max), than they are all applied to all the sensors and multiple results are returned from each sensors. When named list (names are the sensor ids) of functions is provided then mc_agg() apply specific functions to the specific sensors based on the named list list(TMS_T1=c("max", "min"), TMS_T2="mean"). mc_agg returns new sensors on the localities putting aggregation function in its name (TMS_T1 -> TMS_T1_max), despite sensor names contains aggregation function, sensor_id stays the same as before aggregation in sensor metadata (e.g. TMS_T1 -> TMS_T1). Sensors created with functions min, max, mean, percentile, sum, range keeps identical sensor_id and value_type as original input sensors. When function sum is applied on logical sensor (e.g. snow as TRUE, FALSE) the output is integer i.e. number of TRUE values.

Sensors created with functions count has sensor_id count and value_type integer, function coverage has sensor_id coverage and value_type real

Value

Returns new myClim object in Agg-format see myClim-package When fun=NULL, period=NULL records are not modified but only converted to Agg-format. When fun and period are provided then time step is aggregated based on function.

Examples

```
hour_data <- mc_agg(mc_data_example_clean, c("min", "max", "percentile"),
                              "hour", percentiles = 50, min_coverage=0.5)
day_data <- mc_agg(mc_data_example_clean, list(TMS_T1=c("max", "min"), TMS_T2="mean"),
                             "day", min_coverage=1)
month_data <- mc_agg(mc_data_example_clean, fun=list(TMS_T3="below5"),period = "month",
                         custom_functions = list(below5=function(x){length(x[x<(-5)])}))
```

`mc_calc_cumsum` *Cumulative sum*

Description

This function creates new virtual sensor on locality within myClim data object. The virtual sensor hosts the cumulative sum of the values on the input sensor. Names of new sensors are original sensor name + `outpus_suffix`.

Usage

```
mc_calc_cumsum(data, sensors, output_suffix = "_cumsum", localities = NULL)
```

Arguments

`data`	cleaned myClim object see myClim-package
`sensors`	names of sensors where to calculate cumulative sum
`output_suffix`	name suffix for virtual sensor names (default "_cumsum") e.g. TMS_T3_cumsum
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

If value type of sensor is logical, then output type is integer. (TRUE, TRUE, FALSE -> 2)

Value

The same myClim object as input but with added cumsum sensors.

Examples

```
cumsum_data <- mc_calc_cumsum(mc_data_example_agg, c("TMS_T1", "TMS_T2"))
```

mc_calc_fdd *Freezing Degree Days*

Description

This function creates new virtual sensor on locality within myClim data object. The virtual sensor hosts FDD Freezing Degree Days.

Usage

```
mc_calc_fdd(data, sensor, output_prefix = "FDD", t_base = 0, localities = NULL)
```

Arguments

data	cleaned myClim object see myClim-package
sensor	name of temperature sensor used for FDD calculation e.g. TMS_T3 see names(mc_data_sensors)
output_prefix	name prefix of new FDD sensor (default "FDD") name of output sensor consists of output_prefix and value t_base (FDD0_TMS_T3)
t_base	threshold temperature for FDD calculation (default 0)
localities	list of locality_ids for calculation; if NULL then all (default NULL)

Details

The allowed step length for FDD calculation is day and shorter. Function creates new virtual sensor with the same time step as input data. For shorter time steps than the day (which is not intuitive for FDD) the FDD value is the contribution of the time step to the freezing degree day. Be careful while aggregating freezing degree days to longer periods see mc_agg() only meaningful aggregation function is sum, but myClim allow you to apply anything.

Note that FDD is always positive number, despite summing freezing events. When you set t_base=-1 you get the sum of degree days below -1 °C but expressed in positive number if you set t_base=1 you get also positive number. Therefore pay attention to name of output variable which contains t_base value. FDD1_TMS_T3, t_base=1 vs FDDminus1_TMS_T3, t_base=-1

Value

The same myClim object as input but with added virtual FDD sensor

Examples

```
fdd_data <- mc_calc_fdd(mc_data_example_agg, "TMS_T3", localities = c("A2E32", "A6W79"))
fdd_agg <- mc_agg(fdd_data, list(TMS_T3=c("min", "max"), FDD5="sum"), period="day")
```

mc_calc_gdd *Growing Degree Days*

Description

This function creates new virtual sensor on locality within myClim data object. The virtual sensor with values of GDD (Growing Degree Days) is in Celsius degrees in original time step. see details

Usage

```
mc_calc_gdd(data, sensor, output_prefix = "GDD", t_base = 5, localities = NULL)
```

Arguments

data	cleaned myClim object see myClim-package
sensor	name of temperature sensor used for GDD calculation e.g. TMS_T3 see `names(mc_data_sensors)`
output_prefix	name prefix of new GDD sensor (default "GDD" -> "GDD5_TMS_T3") name of output sensor consists of output_prefix and value t_base e.g. GDD5
t_base	base temperature for calculation of GDD (default 5°C)
localities	list of locality_ids for calculation; if NULL then all (default NULL)

Details

The allowed time step length for GDD calculation is day and shorter. Function creates new virtual sensor with the same time step as input data. For shorter time steps than the day, the GDD value is the contribution of the interval to the growing degree day. Be careful while aggregating growing degree days to longer periods see `mc_agg()` only meaningful aggregation function is `sum`, but myClim let you apply anything.

Value

The same myClim object as input but with added virtual GDD sensor

Examples

```
gdd_data <- mc_calc_gdd(mc_data_example_agg, "TMS_T3", localities = c("A2E32", "A6W79"))
gdd_agg <- mc_agg(gdd_data, list(TMS_T3=c("min", "max"), GDD5="sum"), period="day")
```

mc_calc_snow *Snow detection from temperature*

Description

This function creates new virtual sensor on locality within myClim data object. Virtual sensor hosts values of snow cover presence/absence detected from temperature time-series.

Usage

```
mc_calc_snow(
  data,
  sensor,
  output_sensor = "snow",
  localities = NULL,
  range = 1,
  tmax = 1.25,
  days = 3
)
```

Arguments

data	cleaned myClim object see myClim-package
sensor	name of temperature sensor used for snow estimation. (e.g. TMS_T2)
output_sensor	name of output snow sensor (default "snow")
localities	list of locality_ids where snow will be calculated; if NULL then all (default NULL)
range	maximum temperature range threshold for snow-covered sensor (default 1°C)
tmax	maximum temperature threshold for snow-covered sensor (default 1.25°C)
days	number of days to be used for moving-window for snow detection algorithm (default 3 days)

Details

Function detects snow cover from temperature time-series. Temperature sensor is considered as covered by snow when the maximal temperature in the preceding or subsequent time-window (specified by days param) does not exceed specific tmax threshold value (default 1.25°C) and the temperature range remain below specified range threshold (default 1°C). This function rely on insulating effect of a of snow layer, significantly reducing diurnal temperature variation and restricting the maximal temperature near the ground close to freezing point. Temperature sensor near the ground (TMS_T2) is default choice for snow-cover detection from Tomst TMS4 loggers. Snow detection with default values accurately detects snow of depth > 15cm (unpublished data). For detection of thin snow, range parameter should be set to 3-4 °C. The function returns vector of snow cover (TRUE/FLASE) with same time-step as input data. To get number of days with snow cover and more snow summary characteristics use mc_calc_snow_agg after snow detection.

Value

myClim object with added virtual sensor 'snow' (logical) indicating snow presence/absence (TRUE/FALSE).

Examples

```
data <- mc_calc_snow(mc_data_example_agg, "TMS_T2", output_sensor="TMS_T2_snow",
                     localities = c("A2E32", "A6W79"))
```

`mc_calc_snow_agg` *Summary of TRUE/FALSE snow sensor*

Description

This function works with the virtual snow sensor of TRUE/FALSE which is the output of `mc_calc_snow()`. So, before calling `mc_calc_snow_agg` you need to calculate or import `mc_read_` TRUE/FALSE snow sensor. `mc_calc_snow_agg` returns the summary table of snow sensor (e.g number of days with snow cover, first and last date of continual snow cover longer than input period). The snow summary is returned for whole date range provided. And is returned as new data.frame in contrast with other mc_calc functions returning virtual sensors.

Usage

```
mc_calc_snow_agg(
  data,
  snow_sensor = "snow",
  localities = NULL,
  period = 3,
  use_utc = FALSE
)
```

Arguments

`data`	cleaned myClim object see myClim-package with TRUE/FALSE snow sensor see `mc_calc_snow()`
`snow_sensor`	name of snow sensor containing TRUE/FALS snow detection, suitable for virtual sensors created by function `mc_calc_snow`; (default "snow")
`localities`	optional subset of localities where to run the function (list of locality_ids); if NULL then return all localities (default NULL)
`period`	number of days defining the continual snow cover period of interest (default 3 days)
`use_utc`	if set FALSE then time is shifted based on offset provided in locality metadata `tz_offset`, see e.g. `mc_prep_solar_tz()`, `mc_prep_meta_locality()`; (default FALSE)

Details

Primary designed for virtual snow sensor calculated by `mc_calc_snow()`, but accepts any sensor with TRUE/FLAST snow event detection. If `snow_sensor` on the locality is missing, then locality is skipped.

Value

Returns data.frame with columns:

- locality - locality id
- snow_days - number of days with snow cover
- first_day - first day with snow
- last_day - last day with snow
- first_day_period - first day of period with continual snow cover based on `period` parameter
- last_day_period - last day of period with continual snow cover based on `period` parameter

Examples

```
data <- mc_calc_snow(mc_data_example_agg, "TMS_T2", output_sensor="TMS_T2_snow",
                     localities = c("A2E32", "A6W79"))
mc_calc_snow_agg(data, "TMS_T2_snow")
```

`mc_calc_tomst_dendro` *Calibrating Tomst dendrometer values to micrometers*

Description

This function creates new virtual sensor on locality within myClim data object. The virtual sensor hosts the values of the change in stem size converted from raw Tomst units to micrometers. Note that newer versions of Tomst Lolly downloading software can converts raw Tomst units to micrometers.

Usage

```
mc_calc_tomst_dendro(
  data,
  dendro_sensor = .model_const_SENSOR_DEND_TOMSTdendro,
  output_sensor = .model_const_SENSOR_dendro_l_um,
  localities = NULL
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`dendro_sensor`	name of change in stem size sensor to be converted from raw to micrometers (default "DEND_TOMSTdendro") see `names(mc_data_sensors)`
`output_sensor`	name of new change in stem size sensor (default "dendro_l_um")
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Value

myClim object same as input but with added dendro_l_um sensor

Examples

```
agg_data <- mc_calc_tomst_dendro(mc_data_example_agg, localities="A1E05")
```

`mc_calc_vpd` *Calculate vapor pressure deficit (in kPa)*

Description

This function creates new virtual sensor on locality within myClim data object. The virtual sensor hosts the vapor pressure deficit (in kPa) calculated from temperature and relative air humidity.

Usage

```
mc_calc_vpd(
  data,
  temp_sensor = "HOBO_T_C",
  rh_sensor = "HOBO_RH",
  output_sensor = "VPD",
  elevation = 0,
  metadata_elevation = TRUE,
  localities = NULL
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`temp_sensor`	name of temperature sensor. Temperature sensor must be in T_C physical.
`rh_sensor`	name of relative air humidity sensor. Humidity sensor must be in RH_perc physical.
`output_sensor`	name of new virtual VPD sensor (default "VPD")
`elevation`	value in meters (default 0)
`metadata_elevation`	if TRUE then elevation from metadata of locality is used (default TRUE)
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

Equation comes from the CR-5 Users Manual 2009–12 from Buck Research modified from Buck (1981) and adapted by Jones, 2013 (eq. 5.15) Elevation to pressure conversion function uses eq. 3.7 from Campbell G.S. & Norman J.M. (1998).

Value

myClim object same as input but with added VPD sensor

References

Jones H.G. (2014) Plants and Microclimate, Third Edit. Cambridge University Press, Cambridge Buck A.L. (1981) New equations for computing vapor pressure and enhancment factor. Journal of Applied Meteorology 20: 1527–1532. Campbell G.S. & Norman J.M. (1998). An Introduction to Environmental Biophysics, Springer New York, New York, NY

Examples

```
agg_data <- mc_calc_vpd(mc_data_example_agg, "HOBO_T_C", "HOBO_RH", localities="A2E32")
```

---

mc_calc_vwc *Calibrating TMS soil moisture from raw TMS to VWC*

Description

This function creates new virtual sensor on locality within myClim data object. Function converts the soil moisture from raw TMS units (scaled TDT signal) to volumetric water content (VWC).

Usage

```
mc_calc_vwc(
  data,
  moist_sensor = .model_const_SENSOR_TMS_TMSmoisture,
  temp_sensor = .model_const_SENSOR_TMS_T1,
  output_sensor = "VWC_moisture",
  soiltype = "universal",
  localities = NULL,
  ref_t = .calib_MOIST_REF_T,
  acor_t = .calib_MOIST_ACOR_T,
  wcor_t = .calib_MOIST_WCOR_T,
  frozen2NA = TRUE
)
```

Arguments

data	cleaned myClim object see myClim-package
moist_sensor	name of soil moisture sensor to be converted from TMS units to volumetric (default "TMS_TMSmoisture") see names(mc_data_sensors). Soil moisture sensor must be in TMSmoisture physical see names(mc_data_physical).
temp_sensor	name of soil temperature sensor (default "TMS_T1") see names(mc_data_sensors). Temperature sensor must be in T_C physical.
output_sensor	name of new snow virtual sensor with VWC values (default "VWC_moisture")

soiltype	value from mc_data_vwc_parameters in column soiltype (default "universal"). Parameters a, b and c are used in calculation.
localities	list of locality_ids for calculation; if NULL then all (default NULL)
ref_t	(default 24)
acor_t	(default 1.91132689118083) correction temperature while sensor on the air see mc_calib_moisture()
wcor_t	(default 0.64108) correction temperature while sensor in the water mc_calib_moisture()
frozen2NA	if TRUE then those moisture records are set to NA when soil temperature is below 0 (default TRUE)

Details

This function is suitable for Tomst TMS loggers measuring soil moisture in raw TDT units (TMS units). Raw TDT units represents inverted and scaled (1-4095) number of high frequency-shaped electromagnetic pulses (ca 2.5 GHz) sent through a ca 30 cm long circuit within a 640-microsecond time window. For more details see (Wild et al. 2019). Pulse count is directly related to the soil moisture content. Therefore, based on experimental calibration curves, it is possible to directly convert TMS units to standardized volumetric water content in cubic meters. For more details see (Kopecky et al. 2021).

The function uses experimentally derived calibration curves under reference temperatures for several soil types. For the volumetric water content conversion the reference temperature is corrected with the actual soil temperature using TMS_T1 soil temperature sensor records. As the calibration curves were derived for several soil types, in case user know specific soil type, where the logger was measuring, then it is strongly recommended to chose the closest existing calibration curve for specific soil type instead of default "universal". Available soil types are: sand, loamy sand A, loamy sand B, sandy loam A, sandy loam B, loam, silt loam, peat, water, #' universal, sand TMS1, loamy sand TMS1, silt loam TMS1. For more details see (Wild et al. 2019). For full table of function parameters see mc_data_vwc_parameters

The function by default replace the moisture records in frozen soils with NA, because the soil moisture sensor was not designed to measure in frozen soils and the returned records are thus not comparable with values from non-frozen soil.

Value

myClim object same as input but with added virtual VWC moisture sensor

References

Wild, J., Kopecky, M., Macek, M., Sanda, M., Jankovec, J., Haase, T., 2019. Climate at ecologically relevant scales: A new temperature and soil moisture logger for long-term microclimate measurement. Agric. For. Meteorol. 268, 40-47. https://doi.org/10.1016/j.agrformet.2018.12.018

Kopecky, M., Macek, M., Wild, J., 2021. Topographic Wetness Index calculation guidelines based on measured soil moisture and plant species composition. Sci. Total Environ. 757, 143785. https://doi.org/10.1016/j.scitotenv.2020.143785

See Also

mc_data_vwc_parameters

Examples

```
agg_data <- mc_calc_vwc(mc_data_example_agg, soiltype="sand", localities="A2E32")
```

`mc_calib_moisture` *Calculates coefficients for TMS moisture conversion to VWC*

Description

Specialized, service function. You will typically not need to use this one. Function calculate correction parameters (slope and intercept) from TMS moisture measurements in pure water and air.

Usage

```
mc_calib_moisture(
  raw_air,
  raw_water,
  t_air = 24,
  t_water = 24,
  ref_air = 114.534,
  ref_water = 3634.723,
  ref_t = .calib_MOIST_REF_T,
  acor_t = .calib_MOIST_ACOR_T,
  wcor_t = .calib_MOIST_WCOR_T
)
```

Arguments

`raw_air`	Raw TMS moisture records in air
`raw_water`	Raw TMS moisture records in water
`t_air`	temperature of air (default 24)
`t_water`	temperature of water (default 24)
`ref_air`	(default 114.534)
`ref_water`	(default 3634.723)
`ref_t`	(default 24)
`acor_t`	(default 1.91132689118083)
`wcor_t`	(default 0.64108)

Details

This is highly specialized service function designed to derive correction parameters `slope` and `intercept` for soil moisture sensor of TMS loggers measuring on the air and in the water. Slope and intercept calculated in this function could be used as sensor-specific alternative in `mc_calc_vwc()` function instead of defaults. User is not allowed to modify slope and intercept parameters when calling `mc_calc_vwc()`. But this must be done in code of the function. It is possible, but advanced, and typically not necessary, the default values should be working well.

Value

list with slope and intercept parameters

Examples

```
mc_calib_moisture(120, 3650)
```

`mc_DataFormat-class` *Class for logger file data format*

Description

The Class used for parsing source data files. Typically the csv files downloaded from microclimatic loggers. Each supported logger has established own specific object of class `mc_{logger}DataFormat` defining the parameters.

Details

The logger definitions are stored in R environment object `./data/mc_data_formats.rda`. And thus it easy to add the ability of reading new, unsupported logger just by defining its Class parameters. Below see e.g. the Class defining Tomst file format.

```
An object of class "mc_TOMSTDataFormat"
attr(,"skip"): 0
attr(,"separator"): ";"
attr(,"date_column"): 2
attr(,"date_format"): NA
attr(,"na_strings"): "-200"
attr(,"columns"): list()
attr(,"filename_serial_number_pattern"): "data_(\d+)_\d+\.csv$"
attr(,"data_row_pattern"): "^\d+;[\d.: ]+;\d+;-?\d+[.,]?\d*;-?\d+[.,]?\d*;-?\d+[.,]?\d*;\d+;\d+;\d+.
attr(,"logger_type"): character(0)
```

Slots

`skip` number of lines before data - header etc. (default 0)

`separator` columns separator (default NA)

`date_column` index of date column (default NA)

`date_format` format of date (default NA)

`na_strings` strings for NA values (default NA)

`error_value` value means error of sensor (default NA)

`columns` list with names and indexes of value columns (default list())

`filename_serial_number_pattern` character pattern for detecting serial_number from file name (default NA)

`data_row_pattern` character pattern for detecting right file format (default NA)
If data_row_pattern is NA, then file format is not validated.

`logger_type` type of logger: TMS, TMS_L45, ThermoDatalogger, Dendrometer, HOBO, ... (default NA)

`tz_offset` timezone offset in minutes from UTC in source data (default 0)

See Also

mc_data_formats, mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat, mc_HOBODataFormat

---

mc_data_example_agg *Example data in Agg-format.*

Description

Cleaned data in Agg-format. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_TMSmoisture")
- Tomst Thermologger with 1 sensor ("TS_T)
- Tomst Point Dendrometer with 2 sensors ("DEND_T", "DEND_TOMSTdendro")
- HOBO U23 with 2 sensors ("HOBO_T_C", "HOBO_RH")

Usage

```
mc_data_example_agg
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

---

mc_data_example_clean *Example cleaned data in Raw-format.*

Description

Cleaned data. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_TMSmoisture")
- Tomst Thermologger with 1 sensor ("TS_T)
- Tomst Point Dendrometer with 2 sensors ("DEND_T", "DEND_TOMSTdendro")
- HOBO U23 with 2 sensors ("HOBO_T_C", "HOBO_RH")

Usage

```
mc_data_example_clean
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

---

mc_data_example_raw *Example data in Raw-format*

Description

Raw data, not cleaned. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_TMSmoisture")
- Tomst Thermologger with 1 sensor ("TS_T)
- Tomst Point Dendrometer with 2 sensors ("DEND_T", "DEND_TOMSTdendro")
- HOBO U23 with 2 sensors ("HOBO_T_C", "HOBO_RH")

Usage

```
mc_data_example_raw
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

---

mc_data_formats *Formats of source data files*

Description

R object of class environment with the definitions how to parse specific microclimatic logger files. In case you would like to add new, unsupported logger, this is the place where the reading key is stored.

Usage

```
mc_data_formats
```

Format

An object of class `environment` of length 3.

Details

Package myClim support formats TOMST, TOMST_join and HOBO. The environment object is stored in `./data/mc_data_formats.rda`.

TOMST

TOMST data format has stable structure. Datetime is in UTC. Name of data file is in format data_\<serial_number\>_\<x\>.csv. Value serial_number can be automatically detected from file name. Supported loggers are TMS and ThermoDataLogger.

TOMST_join

TOMST_join data format is custom format for internal using of Institute of Botany of the Czech Academy of Sciences. It is the output of joinTMS.exe modified, checked, curated and validated by Lucia.

HOBO

HOBO data format is export format from software HOBOware of Onset company. Format is very variable and can be adjusted by user in preferences of HOBOware. Strucuture of HOBO files format can be partly detected automatically from header of data. Except of format of date-time (`date_format`) which must be set manually in myClim reading functions (`mc_read_files()`, `mc_read_data()`). Date and time separated in more columns is not supported in myClim reading. If time zone is not defined in header of HOBO txt or csv file and is not UTC, then `tz_offset` must be filled in while reading. UTF-8 encoding of HOBO file is required for reding to myClim.

See Also

mc_DataFormat, mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat, mc_HOBODataFormat

---

`mc_data_heights` *Default heights of sensors*

Description

This table is used to set the default heights in metadata of sensors based on logger type. The defaults were set based on the most common uses, defaults can be overwrite be user. see mc_prep_meta_sensor

Usage

```
mc_data_heights
```

Format

An object of class `data.frame` with 13 rows and 4 columns.

Details

data.frame with columns:

- logger_type
- sensor_name
- height - character representation of height
- suffix - suffix for sensor_name. If suffix is NA, then sensor_name is not modified.

Default heights are:

- TS_T = air 200 cm
- TMS_T1 = soil 8 cm
- TMS_T2 = air 2 cm
- TMS_T3 = air 15 cm
- TMS_TMSmoisture = soil 0-15 cm
- DEND_T = 130 cm
- DEND_TOMSTdendro = 130 cm
- HOBO_T_C = air 150 cm
- HOBO_RH = air 150 cm
- TMS_T1 = soil 40 cm
- TMS_T2 = soil 30 cm
- TMS_T3 = air 15 cm
- TMS_TMSmoisture = soil 30-44 cm

See Also

`mc_read_files()`, `mc_read_data()`

`mc_data_physical` *Physical quantities definition*

Description

R object of class environment with the definitions of physical elements for recording the microclimate e.g. temperature, speed, depth, volumetric water content... see mc_Physical. Similarly as in case of logger format definitions mc_DataFormat it is easy to add new, physical here.

Usage

```
mc_data_physical
```

Format

An object of class `environment` of length 12.

See Also

mc_Physical

Currently supported physical elements:

- l_cm - length in cm
- l_mm - length in mm
- l_um - length in um
- moisture - moisture in ratio 0-1
- RH_perc - relative humidity in %
- T_C - temperature in °C
- T_F - temperature in °F
- t_h - time in hours
- TMSmoisture - raw TMS moisture sensor units
- TOMSTdendro - radius difference in raw units
- v - speed in m/s

`mc_data_sensors` *Sensors definition.*

Description

R object of class environment with the definitions of (micro)climatic sensors. see mc_Sensor. Similarly as in case of logger format definitions mc_DataFormat it is easy to add new, sensor here. There is also universal sensor `real` where you can store any real values.

Usage

```
mc_data_sensors
```

Format

An object of class `environment` of length 28.

Details

Names of items are sensor_ids. Currently supported sensors:

- count - result of `count` function `mc_agg()`
- coverage - result of `coverage` function `mc_agg()`
- DEND_T - temperature in Tomst dendrometer (°C)
- DEND_TOMSTdendro - change in stem size in Tomst dendrometer (raw units) `mc_calc_tomst_dendro()`
- dendro_l_um - change in stem size (um) `mc_calc_tomst_dendro()`
- FDD - result of function `mc_calc_fdd()`

- GDD - result of function mc_calc_gdd()
- HOBO_RH - relative humidity in HOBO logger (%)
- HOBO_T_C - temperature in HOBO logger (°C)
- HOBO_T_F - temperature in HOBO logger (°F)
- integer - universal sensor with integer values
- logical - universal sensor with logical values
- moisture - volumetric water content in soil (ratio)
- precipitation - (mm)
- real - universal sensor with real values
- RH_perc - relative humidity sensor (%)
- snow_bool - result of function mc_calc_snow()
- snow_fresh - fresh snow height (cm)
- snow_total - total snow height (cm)
- sun_shine - time of sun shine (hours)
- T_C - universal temperature sensor (°C)
- TS_T - temperature sensor in Tomst Thermologger (°C)
- TMS_T1 - soil temperature sensor in Tomst TMS (°C)
- TMS_T2 - surface temperature sensor in Tomst TMS (°C)
- TMS_T3 - air temperature sensor in Tomst TMS (°C)
- TMS_TMSmoisture - soil moisture sensor in Tomst TMS (raw TMS units)
- wind - wind speed (m/s)

`mc_data_vwc_parameters`

Volumetric water content parameters

Description

Data frame hosting the coefficients for the conversion of TMS raw moisture units to volumetric warer content. The coefficients come from laboratory calibration for several soil types. For the best performance you should specify the soil type in case you know it and in case it could be approximated to the available calibration e.g sand, loam, loamy sand.... See mc_calc_vwc()

Usage

```
mc_data_vwc_parameters
```

Format

An object of class `data.frame` with 13 rows and 9 columns.

Details

data.frame with columns:

- soiltype
- a
- b
- c
- rho
- clay
- silt
- sand
- ref

References

Wild, J., Kopecky, M., Macek, M., Sanda, M., Jankovec, J., Haase, T., 2019. Climate at ecologically relevant scales: A new temperature and soil moisture logger for long-term microclimate measurement. Agric. For. Meteorol. 268, 40-47. https://doi.org/10.1016/j.agrformet.2018.12.018

Kopecky, M., Macek, M., Wild, J., 2021. Topographic Wetness Index calculation guidelines based on measured soil moisture and plant species composition. Sci. Total Environ. 757, 143785. https://doi.org/10.1016/j.scitotenv.2020.143785

`mc_env_moist` *Standardised myClim soil moisture variables*

Description

The wrapper function returning 4 standardised ecologically relevant myClim variables derived from soil moisture. The mc_env_moist function needs time-series of volumetric water content (VWC) measurements as input. Therefore, non-VWC soil moisture measurements must be first converted to VWC. For Tomst loggers see `mc_calc_vwc()`

Usage

```
mc_env_moist(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1
)
```

Arguments

data	cleaned myClim object see myClim-package
period	output period see mc_agg()
use_utc	if FALSE, then local time is used for day aggregation see mc_agg() (default TRUE)
custom_start	start date for custom period see mc_agg() (default NULL)
custom_end	end date for custom period see mc_agg() (default NULL)
min_coverage	the threshold specifying how many missing values can you accept within aggregation period. see mc_agg() value from range 0-1 (default 1)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. In contrast with other myClim functions returning myClim objects this wrapper function return long table. Variables are named based on sensor name, height, and function e.g., (VWC.soil_0_15_cm.5p, VWC.soil_0_15_cm.mean)

Standardised myClim soil moisture variables:

- VWC.5p: Minimum soil moisture = 5th percentile of VWC values
- VWC.mean: Mean soil moisture = mean of VWC values
- VWC.95p: Maximum soil moisture = 95th percentile of VWC values
- VWC.sd: Standard deviation of VWC measurements

Value

table in long format with standardised myClim variables

Examples

```
data <- mc_prep_crop(mc_data_example_agg, lubridate::ymd_h("2020-11-01 00"),
                     lubridate::ymd_h("2021-02-01 00"), end_included = FALSE)
data <- mc_calc_vwc(data, localities=c("A2E32", "A6W79"))
mc_env_moist(data, "month")
```

---

mc_env_temp *Standardised myClim temperature variables*

Description

The wrapper function returning 7 standardised ecologically relevant myClim variables derived from temperature.

Usage

```
mc_env_temp(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1,
  gdd_t_base = 5,
  fdd_t_base = 0
)
```

Arguments

data	cleaned myClim object see myClim-package
period	output period see mc_agg()
use_utc	if FALSE, then local time is used for day aggregation see mc_agg() (default TRUE)
custom_start	start date for custom period see mc_agg() (default NULL)
custom_end	end date for custom period see mc_agg() (default NULL)
min_coverage	the threshold specifying how many missing values can you accept within aggregation period. see mc_agg() value from range 0-1 (default 1)
gdd_t_base	base temperature for Growing Degree Days mc_calc_gdd() (default 5)
fdd_t_base	base temperature for Freezing Degree Days mc_calc_fdd() (default 0)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. It automatically use all available sensors in myClim object and returns all possible variables based on sensor type and measurement height/depth. In contrast with other myClim functions returning myClim objects this wrapper function return long table. The mc_env_temp function first aggregates time-series to daily time-step and then aggregates to the final time-step set in period parameter. Because freezing and growing degree days are always aggregated with sum function, these two variables are not first aggregated to the daily time-steps. Variables are named based on sensor name, height, and function e.g., (T.air_15_cm.max95p, T.air_15_cm.drange)

Standardised myClim temperature variables:

- min5p: Minimum temperature = 5th percentile of daily minimum temperatures
- mean: Mean temperature = mean of daily mean temperatures
- max95p: Maximum temperature = 95th percentile of daily maximum temperatures
- drange: Temperature range = mean of daily temperature range (i.e., difference of daily minima and maxima)
- GDD5: Growing degree days = sum of growing degree days above defined base temperature (default 5°C) gdd_t_base

- FDD0: Freezing degree days = sum of freezing degree days below defined base temperature (default 0°C) `fdd_t_base`
- frostdays: Frost days = number of days with frost (daily minimum < 0°C) `fdd_t_base`

Value

table in long format with standardised myClim variables

Examples

```
data <- mc_prep_crop(mc_data_example_clean, lubridate::ymd_h("2020-11-01 00"),
                     lubridate::ymd_h("2021-02-01 00"), end_included = FALSE)
mc_env_temp(data, "month")
```

`mc_env_vpd` *Standardised myClim vapor pressure deficit variables*

Description

The wrapper function returning 2 standardised ecologically relevant myClim variables derived from vapor pressure deficit. The mc_env_vpd function needs time-series of vapor pressure deficit measurements as input. Therefore, VPD must be first calculated from temperature and air humidity sensors see `mc_calc_vpd()`

Usage

```
mc_env_vpd(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`period`	output period see `mc_agg()`
`use_utc`	if FALSE, then local time is used for day aggregation see `mc_agg()` (default TRUE)
`custom_start`	start date for custom period see `mc_agg()` (default NULL)
`custom_end`	end date for custom period see `mc_agg()` (default NULL)
`min_coverage`	the threshold specifying how many missing values can you accept within aggregation period. see `mc_agg()` value from range 0-1 (default 1)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. The mc_env_vpd function first aggregates time-series to daily time-step and then aggregates to the final time-step set in `period` parameter. In contrast with other myClim functions returning myClim objects this wrapper function return long table. Variables are named based on sensor name, height, and function e.g., (VPD.air_150_cm.mean, VPD.air_150_cm.max95p)

Standardised myClim vapor pressure deficit variables:

- VPD.mean: Mean vapor pressure deficit = mean of daily mean VPD
- VPD.max95p: Maximum vapor pressure deficit = 95th percentile of daily maximum VPD

Value

table in long format with standardised myClim variables

`mc_filter` *Filter data from myClim object*

Description

This function filter data by localities and sensors.

Usage

```
mc_filter(
  data,
  localities = NULL,
  sensors = NULL,
  reverse = FALSE,
  stop_if_empty = TRUE
)
```

Arguments

`data`	myClim object see myClim-package
`localities`	locality_ids for filtering data; if NULL then do nothing
`sensors`	sensor_names for filtering data; if NULL then do nothing see `names(mc_data_sensors)`
`reverse`	if TRUE then input localities and/or sensors are excluded (default FALSE)
`stop_if_empty`	if TRUE then error for empty output (default TRUE)

Details

In default settings it returns the object containing input sensors / localities. When you provide vector of localities e.g. `localities=c("A6W79", "A2E32")` selected localities are filtered with all sensors on those localities. The same as When you provide vector of sensors `sensors=c("TMS_T1", "TMS_T2")`, selected sensors are filtered through all localities. When you combine localities and sensors, then filtering return selected sensors on selected localities.

When `reverse = TRUE` and using only localities parameter then the listed localities are removed. Similarly `reverse = TRUE` with proving only sensors parameter, then the listed sensors are removed from all localities. When using `reverse = TRUE` and combining sensors and localities parameters then selected sensors are removed from selected localities.

Value

filtered myClim object

Examples

```
## keep only "A6W79", "A2E32" localities with all their sensors
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"))

## remove "A6W79", "A2E32" localities and keep all others
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"), reverse=TRUE)

## keep only "TMS_T1", and "TMS_T2" sensors on all localities
filtered_data <- mc_filter(mc_data_example_raw, sensors=c("TMS_T1", "TMS_T2"))

## remove "TMS_T1", and "TMS_T2" sensors from all localities
filtered_data <- mc_filter(mc_data_example_raw, sensors=c("TMS_T1", "TMS_T2"),reverse=TRUE)

## keep only "TMS_T1", and "TMS_T2" sensors on "A6W79", "A2E32" localities
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"),
                           sensors=c("TMS_T1", "TMS_T2"))

## remove "TMS_T1", and "TMS_T2" sensors from "A6W79", "A2E32" localities
## and keep all other sensors and localities
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"),
                           sensors=c("TMS_T1", "TMS_T2"), reverse=TRUE)
```

---

mc_HOBODataFormat-class

Class for reading HOBO logger files

Description

Provides the key for reading the HOBO source files. In which column is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

See Also

mc_DataFormat, mc_data_formats

`mc_info` *Get sensors info table*

Description

This function return data.frame with info about sensors

Usage

```
mc_info(data)
```

Arguments

`data`	myClim object see myClim-package

Value

data.frame with columns:

- locality_id - when provided by user then locality ID, when not provided identical with serial number
- serial_number - serial number of logger when provided or automatically detected from file name or header
- sensor_id - original sensor id (e.g.,"GDD", "HOBO_T_C" ,"TMS_T1", "TMS_T2")
- sensor_name - original sensor id if not modified, if renamed then new name (e.g.,"GDD5", "HOBO_T_C_mean" ,"TMS_T1_max", "my_sensor01")
- start_date - the oldest record on the sensor
- end_date - the newest record on the sensor
- step - time step of records series (seconds)
- period - time step of records series (text)
- min_value - minimal recorded values
- max_value - maximal recorded value
- count_values - number of non NA records
- count_na - number of NA records

Examples

```
mc_info(mc_data_example_agg)
```

`mc_info_clean` *Call cleaning log*

Description

This function return data.frame with information from cleaning the loggers time series see `mc_prep_clean()`

Usage

```
mc_info_clean(data)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package

Value

data.frame with columns:

- locality_id - when provided by user then locality ID, when not provided identical with serial number
- serial_number - serial number of logger when provided or automatically detected from file name or header
- start_date - date of the first record on the logger
- end_date - date of the last record on the logger
- step - detected time step in seconds of the logger measurements.
- count_duplicities - number of duplicated records (identical time)
- count_missing - number of missing records (logger outage in time when it should record)
- count_disordered - number of records incorrectly ordered in time (newer followed by older)
- rounded - T/F indication whether myClim automatically rounded time series minutes to the closes half (HH:00, HH:30) e.g. 13:07 -> 13:00

See Also

`mc_prep_clean()`

mc_info_count *Count data*

Description

This function return data.frame with the number of localities, loggers and sensors of input myClim object.

Usage

```
mc_info_count(data)
```

Arguments

data	myClim object see myClim-package

Value

data.frame with count of localities, loggers and sensors

Examples

```
count_table <- mc_info_count(mc_data_example_raw)
```

mc_info_meta *Get localities metadata table*

Description

This function return data.frame with localities metadata

Usage

```
mc_info_meta(data)
```

Arguments

data	myClim object see myClim-package

Value

data.frame with columns:

- locality_id
- lon_wgs84
- lat_wgs84
- elevation
- tz_offset

Examples

```
mc_info_meta(mc_data_example_agg)
```

`mc_join` *Joining time-series from repeated downloads*

Description

Function join time-series from repeated download on the same locality. If time-series overlaps and values are identical or first and last records fit, then automatically joined without any user action. If values are different on overlap, then user interactively selects which source logger to use. Description of conflict and interactive line plots with conflict intervals are shown during joining of each logger one-by-one.

Usage

```
mc_join(data, comp_sensors = NULL)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`comp_sensors`	senors for compare and select source logger; If NULL then first is used. (default NULL)

Details

Joining is semi-automatic pairwise process. In case the end of older and start of the newer time-series fits myClim is sequentially adding the newer time-series to the older without any user action. In case the time series overlap and values on the overlap are not identical, user interactively selects which time series to use or type in the specific datetime where to trim older time-series and continue newer. To support joining interactive lines and text description of conflict time-series are shown. User can zoom in/out the plot and with mouse identify specific datetime where to trim older series and continue newer.

Loggers with multiple sensors are joined base on one or more selected sensors. See parameter `comp_sensors`.

Name of resulting joined sensor is used from logger with the oldest data. If `serial_number` is not equal while joining loggers, then the resulting `serial_number` is `NA`. Clean info is changed to `NA` except step. In case you are joining not calibrated sensor with calibrated one, then calibration information must be empty in not calibrated sensor.

Value

myClim object with joined loggers.

mc_load *Load myClim object*

Description

This function loads the myClim .rds data object saved with mc_save. The `mc_save` and `mc_load` functions secure that the myClim object is correctly loaded across myClim versions.

Usage

```
mc_load(file)
```

Arguments

`file`	path to input .rds file

Value

loaded myClim object

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir)
mc_save(mc_data_example_agg, tmp_file)
data <- mc_load(tmp_file)
file.remove(tmp_file)
```

mc_LocalityMetadata-class *Class for locality metadata*

Description

Class for locality metadata

Details

When reading without metadata, then locality is named after file where the data come from, or after the sensor id where the data come form.

Slots

`locality_id` name of locality

`elevation` of locality

`lat_wgs84` latitude of locality in WGS-84

`lon_wgs84` longitude of locality in WGS-84

`tz_offset` offset from UTC in minutes

`tz_type` type of time zone

`user_data` list for user data

See Also

myClim-package, mc_LoggerMetadata, mc_SensorMetadata

`mc_LoggerCleanInfo-class`
Class for logger clean info

Description

Class for logger clean info

Slots

`step` Time step of microclimatic data series in seconds

`count_duplicities` count of duplicated records - values with same date

`count_missing` count of missing records; Period between the records should be the same length. If not, than missing.

`count_disordered` count of records incorrectly ordered in time. In table, newer record is followed by the older.

`rounded` T/F indication whether myClim automatically rounded time series to the closes half (HH:00, HH:30) e.g. 13:07 -> 13:00

mc_LoggerMetadata-class *Class for logger metadata*

Description

Class for logger metadata

Slots

`type` of logger (TMS, ThermoDatalogger, Dendrometer, HOBO)

`serial_number` serial number of the logger

`step` time step of microclimatic time-seris in seconds. When provided by user, is used in `mc_prep_clean()` function instead of automatic step detection

mc_MainMetadata-class *Class for myClim object metadata*

Description

Class for myClim object metadata

Slots

`version` the version of the myClim package in which the object was created

`format_type` type of format (Raw-format, Agg-format)

See Also

myClim-package

mc_MainMetadataAgg-class *Class for myClim object metadata in Agg-format*

Description

Class for myClim object metadata in Agg-format

Slots

`version` the version of the myClim package in which the object was created

`format_type` type of format (Raw-format, Agg-format)

`step` time step of data in seconds

`period` value from `mc_agg()` (e.g. month, day, all...)

`intervals_start` start datetime of data intervals for spacial periods all and custom (see `mc_agg()`)

`intervals_end` end datetime of data intervals for spacial periods all and custom (see `mc_agg()`)

See Also

mc_MainMetadata myClim-package

mc_Physical-class *Class for physical*

Description

Class defining the element of the records (temperature, volumetric water content, height...)

Details

See e.g. definition of temperature. Similarly as the definition of new loggers, new physicals cn be added like modules.

```
Slot "name": "T_C"
slot "description": "Temperature °C"
Slot "units": "°C"
Slot "viridis_color_map": "C"
Slot "scale_coeff": 0.03333333
```

Slots

`name` of physical

`description` character info

`units` measurument (°C, %, m3/m3, raw, mm, ...)

`viridis_color_map` viridis color map option

`scale_coeff` coefficient for plot; value * scale_coef is in range 0-1

See Also

mc_data_physical

`mc_plot_image` *Plot data - image*

Description

Function plots single sensor form myClim data into PNG file with image() R base function. This was designed for fast, and easy data visualization especially focusing on missing values visualization and general data picture.

Usage

```
mc_plot_image(
  data,
  filename,
  title = "",
  localities = NULL,
  sensors = NULL,
  height = 1900,
  left_margin = 12
)
```

Arguments

`data`	myClim object see myClim-package
`filename`	output file name (file path)
`title`	of plot; default is empty
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`height`	of image; default = 1900
`left_margin`	width of space for sensor_labels; default = 12

Details

Be careful with bigger data. Can take some time.

Value

PNG file created as specified in output file name

mc_plot_line *Plot data - ggplot2 geom_line*

Description

Function plots data with ggplot2 geom_line. Plot is returned as ggplot faced grid and is primary designed to be saved as PDF file. PNG is also possible, the same as plotting ggplot object into R environment. See details.

Usage

```
mc_plot_line(
  data,
  filename = NULL,
  sensors = NULL,
  scale_coeff = NULL,
  png_width = 1900,
  png_height = 1900,
  start_crop = NULL,
  end_crop = NULL
)
```

Arguments

data	myClim object see myClim-package
filename	output file name/path with the extension - supported formats are .pdf and .png (default NULL) If NULL then the plot is displayed and can be returned into r environment but is not saved to file.
sensors	select the names of sensors to be plotted (max 2) see `names(mc_data_sensors)`
scale_coeff	scale coefficient for secondary axis (default NULL)
png_width	width for png output (default 1900)
png_height	height for png output (default 1900)
start_crop	POSIXct datetime for crop data (default NULL)
end_crop	POSIXct datetime for crop data (default NULL)

Details

Saving as the PDF file is recommended, because the plot is optimized to be paginate PDF (facet raster plot is distributed to pages), which is especially useful for bigger data. Maximal number of physical units (elements) of sensors to be plotted in one plot is two with main and secondary y axis. In case, there are multiple sensors with identical physical on one locality, they are plotted together. E.g., when you have TMS_T1, TMS_T2, TMS_T3, TS_T, and moisture you get plot with 5 lines of different colors and two y axes. Secondary y axes are scaled with calculation `values * scale_coeff`. If coefficient is NULL than function try detects scale coefficient from physical unit

of sensors see mc_Physical. Scaling is useful when plotting together e.g. temperature and moisture. For TMS and HOBO scaling coefficients are estimated automatically, correctly. For other data it is better to set it by hand.

Value

ggplot2 object

Examples

```
tms.plot <- mc_filter(mc_data_example_agg, localities = "A6W79")
p <- mc_plot_line(tms.plot,sensors = c("TMS_T3","TMS_T1","TMS_TMSmoisture"))
p <- p+ggplot2::scale_x_datetime(date_breaks = "1 week", date_labels = "%W")
p <- p+ggplot2::xlab("week")
p <- p+ggplot2::scale_color_manual(values=c("hotpink","pink", "darkblue"),name=NULL)
```

---

mc_plot_loggers *Plot data from loggers*

Description

Function save separate files (*.png) per the loggers to the directory. Only Raw-format supported, Agg-format not supported. For Agg-format use `mc_plot_line()`. Function was primary designed for Tomst TMS loggers for fast, and easy data visualization.

Usage

```
mc_plot_loggers(
  data,
  directory,
  localities = NULL,
  sensors = NULL,
  crop = c(NA, NA)
)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`directory`	path to output directory
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`crop`	datetime range for plot, not cropping if NA (default c(NA, NA))

Value

PNG files created in the output directory

Examples

```
tmp_dir <- file.path(tempdir(), "plot")
mc_plot_loggers(mc_data_example_clean, tmp_dir)
unlink(tmp_dir, recursive=TRUE)
```

`mc_plot_raster` *Plot data - ggplot2 geom_raster*

Description

Function plots data with ggplot2 geom_raster. Plot is returned as ggplot faced raster and is primary designed to be saved as .pdf file (recommended) or .png file. Plotting into R environment without saving any file is also possible. See details.

Usage

```
mc_plot_raster(
  data,
  filename = NULL,
  sensors = NULL,
  by_hour = TRUE,
  png_width = 1900,
  png_height = 1900,
  viridis_color_map = NULL,
  start_crop = NULL,
  end_crop = NULL
)
```

Arguments

`data`	myClim object see myClim-package
`filename`	output with the extension - supported formats are .pdf and .png (default NULL) If NULL then the plot is shown/returned into R environment as ggplot object, but not saved to file.
`sensors`	names of sensor; should have same physical unit see `names(mc_data_sensors)`
`by_hour`	if TRUE, then y axis is plotted as an hour, else original time step (default TRUE)
`png_width`	width for png output (default 1900)
`png_height`	height for png output (default 1900)
`viridis_color_map`	viridis color map option; if NULL, then used value from mc_data_physical • "A" - magma • "B" - inferno • "C" - plasma • "D" - viridis

- "E" - cividis
- "F" - rocket
- "G" - mako
- "H" - turbo

`start_crop` POSIXct datetime for crop data (default NULL)

`end_crop` POSIXct datetime for crop data (default NULL)

Details

Saving as the .pdf file is recommended, because the plot is optimized to be paginate PDF (facet raster plot is distributed to pages), which is especially useful for bigger data. In case of plotting multiple sensors to PDF, the facet grids are grouped by sensor. I.e., all localities of sensor_1 followed by all localities of sensor_2 etc. When plotting only few localities, but multiple sensors, each sensor has own page. I.e., when plotting data from one locality, and 3 sensors resulting PDF has 3 pages. In case of plotting PNG, sensors are plotted in separated images (PNG files) by physical. I.e, when plotting 3 sensors in PNG it will save 3 PNG files named after sensors. Be careful with bigger data in PNG. Play with `png_height` and `png_width`. When too small height/width, image does not fit and is plotted incorrectly. Plotting into R environment instead of saving PDF or PNG is possible, but is recommended only for low number of localities (e.g. up to 10), because high number of localities plotted in R environment results in very small picture which is hard/impossible to read.

Value

list of ggplot2 objects

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir, fileext=".pdf")
mc_plot_raster(mc_data_example_agg, filename=tmp_file, sensors=c("TMS_T3","TM_T"))
file.remove(tmp_file)
```

---

mc_prep_calib *Sensors calibration*

Description

This function calibrate values of sensor (microclimatic records) using the myClim object `sensor$calibration` parameters provided by `mc_prep_calib_load()`. Microclimatic records are changed and myClim object parameter `sensor$metadata@calibrated` is set to TRUE. It isn't allowed to calibrate sensor multiple times.

Usage

```
mc_prep_calib(data, localities = NULL, sensors = NULL)
```

Arguments

data	myClim object in Raw-format or Agg-format having calibration data in metadata slot sensor$calibration
localities	vector of locality_ids where to perform calibration, if NULL, then calibrate sensors on all localities (default NULL)
sensors	vector of sensor names where to perform calibration see names(mc_data_sensors); if NULL, then calibrate all sensors having calibration parameters loaded (default NULL)

Details

This function performs calibration itself. It uses the calibration values (cor_factor, cor_slope) stored in myClim object sensor metadata sensor calibration loaded with mc_prep_calib_load(). As it is possible to have multiple calibration values for one sensor in time (re-calibration after some time) different calibration values can be applied based on the calibration time. Older microclimatic records then first calibration datetime available are calibrated anyway (in case sensor was calibrated ex-post) with the first calibration parameters available.

This function is not designed for TMSmoisture calibration (conversion to volumetric water content) for this use mc_calc_vwc()

Only sensors with real value type can be calibrated. see mc_data_sensors()

Value

same myClim object as input but with calibrated sensor values.

---

mc_prep_calib_load *Load calibration to correct microclimatic records*

Description

This function loads calibration parameters from data.frame and writes them into myClim object metadata. This function does not calibrate data. For calibration itself run mc_prep_calib()

Usage

```
mc_prep_calib_load(data, calib_table)
```

Arguments

data	myClim object in Raw-format. see myClim-package
calib_table	data.frame with columns (serial_number, sensor_id, datetime, slope, intercept)

Details

This function allows user to provide calibration values either from DIY or certified calibration procedure. Calibration data have by default the form of linear function determined by the `cor_factor` and `cor_slope`:

`calibrated = original * (cor_slope + 1) + cor_factor`

This is useful in case of multi-point calibration typically performed by certified calibration labs. In case of one-point calibration typically DIY calibrations only `cor_factor` is used and `cor_slope=0`. One point calibration is thus only the addition of correction factor. This function loads sensor specific calibration values from data frame and writes them into myClim Raw-format object metadata. The structure of input data frame is as follows:

- serial_number = unique identification of logger hosting the sensors e.g. 91184101
- sensor_id = the name of sensor to calibrate e.g. TMS_T1
- datetime = the date of the calibration
- cor_factor = the correction factor, in case of multi-point calibration the intercept of calibration curve.
- cor_slope = the slope of calibration curve (in case of one-point calibration slope = 0)

It is not possible to change calibration parameters for already calibrated sensor. This prevents repeated calibrations. Once `mc_prep_calib()` is called then it is not allowed to provide new calibration data, neither run calibration again.

Value

myClim object with loaded calibration information in metadata. Microclimatic records are not calibrated, only ready for calibration. To calibrate records run `mc_prep_calib()`

`mc_prep_clean` *Cleaning datetime series*

Description

By default `mc_prep_clean` runs automatically when `mc_read_files()`, `mc_read_data()` are called. `mc_prep_clean` check time-series in myClim object in Raw-format for missing, duplicated, and disordered records and regularize microclimatic time-series to constant time-step. Duplicated records are removed and missing values are filled with NA.

See details.

Usage

```
mc_prep_clean(data, silent = FALSE)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`silent`	if true, then cleaning log table is not printed in console (default FALSE), see `mc_info_clean()`

Details

Processing the data with `mc_prep_clean` is a mandatory step required for further data handling in `myClim` library.

This function guarantee that all time series are in chronological order and have regular time-step, without duplicated records. Function `mc_prep_clean` use time-step provided by user on import `mc_read` (is stored in metadata of logger mc_LoggerMetadata. If time step is not provided by user on import (NA),than myClim automatically identify the time step from input time series based on last 100 records. In case of irregular time series, function returns warning and skip series.

In case the time step is regular, but is not nicely rounded, function round the time series to the closest nice time and shift original data. E.g., original records in 10 min regular step c(11:58, 12:08, 12:18, 12:28) are shifted to newly generated nice sequence c(12:00, 12:10, 12:20, 12:30) microclimatic records are not modified but only shifted. Maximal allowed shift of time series is 30 minutes. I.e. when the time step is 2h and goes like (13:33, 15:33, 17:33) then shifted to (13:30, 15:30, 17:30). When you have 2h time step and wish to round to the whole hour (13:33 -> 14:00, 15:33 -> 16:00) than after clening use `mc_agg(period="2 hours")`

Value

- cleaned myClim object in Raw-format
- cleaning log is by default printed in console, and can be called ex post by `mc_info_clean()`

Examples

```
cleaned_data <- mc_prep_clean(mc_data_example_raw)
```

`mc_prep_crop` *Crop datetime*

Description

This function crop data by datetime

Usage

```
mc_prep_crop(data, start = NULL, end = NULL, end_included = TRUE)
```

Arguments

`data`	myClim object see myClim-package
`start`	POSIXct datetime in UTC; is optional; start datetime is included
`end`	POSIXct datetime in UTC; is optional
`end_included`	if TRUE then end datetime is included (default TRUE)

Details

Function is able to crop data from start to end but works also with only start and only end. When only start provided, then crop only start and do not touch the end and vice versa.

Value

cropped data in the same myClim format as input.

Examples

```
cropped_data <- mc_prep_crop(mc_data_example_clean, end=as.POSIXct("2020-02-01", tz="UTC"))
```

`mc_prep_fillNA` *Fill NA*

Description

This function approximate NA (missing) values. It was designed to fill only small gaps in microclimatic time-series therefore, the default maximum length of the gap is 5 missing records and longer gaps are not filled Only linear method is implemented from zoo::na.approx function.

Usage

```
mc_prep_fillNA(
  data,
  localities = NULL,
  sensors = NULL,
  maxgap = 5,
  method = "linear"
)
```

Arguments

`data`	myClim object see myClim-package
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`maxgap`	maximum number of consecutively NA values to fill (default 5)
`method`	used for approximation. It is implemented now only "linear". (default "linear")

Value

myClim object with filled NA values

mc_prep_merge *Merge myClim objects*

Description

This function is designed to merge more existing myClim objects into one.

Usage

```
mc_prep_merge(data_items)
```

Arguments

data_items	list of myClim objects see myClim-package; Format (Raw/Agg) of merged objects must be same.

Details

This function works only when the input myClim objects have the same format (Raw-format, Agg-format) It is not possible to merge Raw wit Agg format. Identical time-step is required for Agg-format data.

When the merged myClim objects in Raw-format contains locality with same names (locality_id), than list of loggers are merged on the locality. Sensors with the same name does not matter here. Loggers with the same name within the locality are allowed in the Raw-format.

When the merged myClim objects in Agg-format contains locality with same names (locality_id). than the sensors are merged on the locality. Sensors with same names are renamed.

Value

merged myClim object in the same format as input objects

Examples

```
merged_data <- mc_prep_merge(list(mc_data_example_raw, mc_data_example_raw))
```

mc_prep_meta_locality *Set metadata of localities*

Description

This function allows you to add or modify locality metadata including locality names. See mc_LocalityMetadata. You can import metadata from named list or from data frame. See details.

Usage

```
mc_prep_meta_locality(data, values, param_name = NULL)
```

Arguments

data	myClim object see myClim-package
values	for localities can be named list or table • named list: metadata <- list(locality_id=value); param_name must be set • table with column locality_id and another columns named by metadata parameter name; to rename locality use new_locality_id. Parameter param_name must be NULL.
param_name	name of locality metadata parameter; Default names are locality_id, elevation, lat_wgs84, lon_wgs84, tz_offset. Another names are inserted to user_data list. see mc_LocalityMetadata

Details

Locality metadata is especially useful for handling time zones, considering temporal cycling. E.g. while providing coordinates, and in case you are sure, the loggers recorded in UTC, you can harmonize all data to the solar time (midday) with mc_prep_solar_tz() calculating offset to the UTC based on coordinates. Or you can directly provide the offset in minutes yourself for individual localities. This is useful e.g. for heterogeneous data sets containing loggers recording in local time and you wish to unify them by setting individual offset e.g. back to UTC. If tz_offset is set manually, than tz_type is set to user defined.

For minor metadata modification it is practical to use named list in combination with param_name specification. E.g. when you wish to modify only time zone offset, then set param_name="tz_offset" and provide named list with locality name and offset value list(A1E05=60). Similarly for other metadata slots mc_LocalityMetadata.

For batch or generally more complex metadata modification you can provide data.frame with columns specifying locality_id and one of new_locality_id, elevation, lat_wgs84, lon_wgs84, tz_offset. Provide locality_id (name) and the value in column of metadata you wish to update. In case of using data.frame use param_name = NULL

Value

myClim object in the same format as input, with updated metadata

Examples

```
data <- mc_prep_meta_locality(mc_data_example_raw, list(A1E05=60), param_name="tz_offset")
```

---

mc_prep_meta_sensor *Set metadata of sensors*

Description

This function allows you to modify sensor metadata including sensor name. See mc_SensorMetadata

Usage

```
mc_prep_meta_sensor(
  data,
  values,
  param_name,
  localities = NULL,
  logger_types = NULL
)
```

Arguments

data	myClim object see myClim-package
values	named list with metadata values; names of items are sensor_names e.g. for changing sensor height use `list(TMS_T1="soil 8 cm")`
param_name	name of the sensor metadata parameter you want to change; You can change `name` and `height` of sensor.
localities	optional filter; vector of `locality_id` where to change sensor metadata; if NULL than all localities (default NULL)
logger_types	optional filter; vector of `logger_type` where to change metadata; if NULL than all logger types (default NULL); `logger_type`is useful only for Raw-format of myClim having the level of logger see myClim-package

Value

myClim object in the same format as input, with updated sensor metadata

Examples

```
data <- mc_prep_meta_sensor(mc_data_example_raw, list(TMS_T1="my_TMS_T1"), param_name="name")
```

---

mc_prep_solar_tz *Set solar time offset against UTC time*

Description

This function calculates the offset against UTC on the locality to get the solar time. This is based on coordinates (longitude). If longitude is not provided, then not working.

Usage

```
mc_prep_solar_tz(data)
```

Arguments

data	myClim object see myClim-package

Details

myClim library presumes the data in UTC by default. This function require at least longitude provided in locality metadata slot `lon_wgs84`. If longitude is not provided, function does not work. Coordinates of locality can be provided e.g. during data reading see `mc_read_data()` or ex post with `mc_prep_meta_locality()` function.

TZ offset in minutes is calculated as `longitude / 180 * 12 * 60`.

Value

myClim object in the same format as input, with `tz_offset` filled in locality metadata

Examples

```
data_solar <- mc_prep_solar_tz(mc_data_example_clean)
```

`mc_read_data` *Reading files with locality metadata*

Description

This function has two tables as the parameters.

(i) `files_table` with *paths* pointing to raw csv logger files, specification of *data format* (logger type) and *locality name*.

(ii) `localities_table` with locality id and metadata e.g. longitude, latitude, elevation...

Usage

```
mc_read_data(
  files_table,
  localities_table = NULL,
  clean = TRUE,
  silent = FALSE
)
```

Arguments

`files_table` path to csv file or data.frame object with 3 required columns and few optional: required columns:

- path - path to files
- locality_id - the unique locality id
- data_format see mc_data_formats, `names(mc_data_formats)`

optional columns:

- serial_number - can be NA, than myClim tries to detect from file name or header

- logger_type - type of logger. This is used to set default sensor heights important when joining the data. In some cases myClim detects loger_type from source data file, but sometimes it is not possible. Pre-defined logger types are: ("Dendrometer","HOBO","ThermoDatalogger","TMS","TMS_L45") Default heights of sensor based on logger types are defined in table mc_data_heights
- date_format - for reading HOBO format of date in strptime function (e.g. "%d.%m.%y %H:%M:%S"); Ignored for Tomst TMS data format
- tz_offset - If source datetimes aren't in UTC, then is possible define offset from UTC in minutes. Value in this column have the highest priority. If NA then auto detection of timezone in files. If timezone can't be detected, then UTC is supposed. Timezone offset in HOBO format can be defined in header. In this case function try detect offset automatically. Ignored for Tomst TMS data format (they are always in UTC)
- step - Time step of microclimatic time-series in seconds. When provided, then used in mc_prep_clean instead of automatic step detection.

`localities_table`
: path to csv file or data.frame. Localities table is optional (default NULL). object containing 5 columns:
  - locality_id
  - elevation
  - lon_wgs84
  - lat_wgs84
  - tz_offset

`clean`
: if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)

`silent`
: if TRUE, then any information is not printed in console (default FALSE)

Details

The input tables could be R data.frames or csv files. When loading `files_table` and `localities_table` from external CSV they must have header, column separator must be comma ",". By default data are cleaned with function `mc_prep_clean()`. See function description. It detects holes in time-series, duplicated records or records in wrong order.

Value

myClim object in Raw-format see myClim-package

See Also

mc_DataFormat

Examples

```
files_csv <- system.file("extdata", "files_table.csv", package = "myClim")
localities_csv <- system.file("extdata", "localities_table.csv", package = "myClim")
tomst_data <- mc_read_data(files_csv, localities_csv)
```

`mc_read_files` *Reading files or directories*

Description

This function read one or more csv files or directories of identical, pre-defined logger type (format) see mc_DataFormat and mc_data_formats. This function does not support loading locality or sensor metadata while reading. Metadata can be loaded through `mc_read_data()` or can be provided later with function `mc_prep_meta_locality()`

Usage

```
mc_read_files(
  paths,
  dataformat_name,
  logger_type = NA_character_,
  recursive = TRUE,
  date_format = NA_character_,
  tz_offset = NA_integer_,
  step = NA_integer_,
  clean = TRUE,
  silent = FALSE
)
```

Arguments

`paths`	vector of paths to files or directories
`dataformat_name`	data format of logger one of `names(mc_data_formats)`
`logger_type`	type of logger (default NA), can be one of pre-defined see `mc_read_data()` or any custom string
`recursive`	recursive search in sub-directories (default TRUE)
`date_format`	format of date in your hobo files e.g. "%d.%m.%y %H:%M:%S" (default NA). Required for HOBO files. For TMS files ignored, there is known, stable date format. see mc_data_formats
`tz_offset`	timezone offset in minutes; It is required only for non-UTC data (custom settings in HOBO). Not used in TMS (default NA)
`step`	time step of microclimatic time-series in seconds. When provided, then is used in mc_prep_clean instead of automatic step detection. If not provided (NA), is automatically detected in mc_prep_clean. (default NA)
`clean`	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
`silent`	if TRUE, then any information is printed in console (default FALSE)

Details

If file is not in expected format, then file is skipped and warning printed in console. CSV files (loggers raw data) are in resulting myClim object placed to separate localities with empty metadata. Localities are named after serial_number of logger. Pre-defined logger types are ("Dendrometer","HOBO","ThermoDatalogger","TMS","TMS_L45") By default data are cleaned with function `mc_prep_clean()`. See function description. It detects holes in time-series, duplicated records or records in wrong order.

Value

myClim object in Raw-format see myClim-package

See Also

mc_DataFormat, `mc_prep_clean()`

Examples

```
files <- c(system.file("extdata", "data_91184101_0.csv", package = "myClim"),
           system.file("extdata", "data_94184102_0.csv", package = "myClim"))
tomst_data <- mc_read_files(files, "TOMST")
```

---

mc_read_long *Reading data from long data.frame*

Description

This is universal function designed to read time series and values from long data.frame to myClim object.

Usage

```
mc_read_long(data_table, sensor_ids = list(), clean = TRUE, silent = FALSE)
```

Arguments

`data_table`	long data.frame with Columns: • locality_id - character; id of locality • sensor_name - can be any character string, recommended are these: `names(mc_data_sensors)` • datetime - POSIXct in UTC timezone is required • value
`sensor_ids`	list with relations between sensor_names and sensor_ids (default list()); sensor_id is key from `names(mc_data_sensors)`. E.g., `sensor_ids <- list(precipitation="real", maxAirT="T_C")` If sensor_name is the same as sensor_id does not have to be provided.
`clean`	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
`silent`	if TRUE, then any information is not printed in console (default FALSE)

Details

Similar like mc_read_wide but is capable to read multiple sensors from single table. Useful for data not coming from supported microclimatic loggers. E.g. meteorological station data. By default data are cleaned with function `mc_prep_clean()`.

Value

myClim object in Raw-format

See Also

mc_read_wide

`mc_read_tubedb` *Reading data from TubeDB*

Description

Function is reading data from TubeDB (https://environmentalinformatics-marburg.github.io/tubedb/) into myClim object.

Usage

```
mc_read_tubedb(
  tubedb,
  region = NULL,
  plot = NULL,
  sensor_ids = NULL,
  clean = TRUE,
  silent = FALSE,
  aggregation = "raw",
  quality = "no",
  ...
)
```

Arguments

`tubedb`	object for connection to server see rTubeDB::TubeDB
`region`	vector of TubeDB region ids - see rTubeDB::query_regions (default NULL) Regions are used mainly for loading metadata from TubeDB localities.
`plot`	vector of localities ids see rTubeDB::query_region_plots rTubeDB::query_timeseries (default NULL) If plot is NULL, then all localities are loaded from whole region.
`sensor_ids`	list in format `list(tubedb_sensor_name=myClim_sensor_name)` (default NULL) If sensor names in TubeDB match the default sensor names in myClim, then the value is detected automatically.

clean	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
silent	if TRUE, then any information is not printed in console (default FALSE)
aggregation	parameter used in function rTubeDB::query_timeseries (default raw)
quality	parameter used in function rTubeDB::query_timeseries (default no)
...	other parameters from function rTubeDB::query_timeseries

Details

In case you store your microclimatic time-series in TubeDB, you can read data with TubeDB API into myClim object. You need to know database URL, username and password.

Value

myClim object in Raw-format

Examples

```
# Not run: To retrieve data from TubeDB, a running TubeDB server with a user account
#          and a secret password is required.
## Not run:
tubedb <- TubeDB(url="server", user="user", password="password")
data <- mc_read_tubedb(tubedb, region="ckras", plot=c("TP_KAR_19", "TP_KODA_61"))

## End(Not run)
```

---

mc_read_wide *Reading data from wide data.frame*

Description

This is universal function designed to read time-series and values from wide data.frame to myClim object. Useful for data not coming from supported microclimatic loggers. E.g. meteorological station data.

Usage

```
mc_read_wide(
  data_table,
  sensor_id = .model_const_SENSOR_real,
  sensor_name = NULL,
  clean = TRUE,
  silent = FALSE
)
```

Arguments

`data_table`	data.frame with first column of POSIXct time format UTC timezone, followed by columns with (micro)climatic records. See details. Columns: • datetime column - POSIXct in UTC timezone is required • Name of locality[1] - values • ... • Name of locality[n] - values
`sensor_id`	define the sensor type, one of `names(mc_data_sensors)` (default `real`)
`sensor_name`	custom name of sensor; if NULL (default) than `sensor_name == sensor_id`
`clean`	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
`silent`	if TRUE, then any information is printed in console (default FALSE)

Details

The first column of input data.frame must be datetime column in POSIXct time format UTC timezone. Following columns represents localities. Column names are the localities names. All values in wide data.frame represents the same sensor type, e.g. air temperature. If you wish to read multiple sensors use mc_read_long or use mc_read_wide multiple times separately for each sensor type and that merge myClim objects with mc_prep_merge By default data are cleaned with function `mc_prep_clean()`. See function description. It detects holes in time-series, duplicated records or records in wrong order.

Value

myClim object in Raw-format

See Also

mc_read_long

`mc_reshape_long` *Export values to long table*

Description

This function converts myClim object to long R data.frame.

Usage

```
mc_reshape_long(data, localities = NULL, sensors = NULL)
```

Arguments

data	myClim object see myClim-package
localities	names of localities; if NULL then all (default NULL)
sensors	names of sensors; if NULL then all (default NULL) see names(mc_data_sensors)

Value

data.frame

columns:

- locality_id
- serial_number
- sensor_name
- height
- datetime
- time_to
- value

Examples

```
head(mc_reshape_long(mc_data_example_clean, c("A6W79", "A2E32"), c("TMS_T1", "TMS_T2")), 10)
```

---

mc_reshape_wide *Export values to wide table*

Description

This function converts myClim object to the R data.frame with values of sensor in wide format.

Usage

```
mc_reshape_wide(data, localities = NULL, sensors = NULL)
```

Arguments

data	myClim object see myClim-package
localities	names of localities; if NULL then all (default NULL)
sensors	names of sensors; if NULL then all (default NULL) see names(mc_data_sensors)

Details

First column of the output data.frame is datetime followed by the columns for every sensor. Name of the column is in format:

- localityid_serialnumber_sensorname for Raw-format
- localityid_sensorname for Agg-format

The less complex wide table is returned when exporting single sensor ascross localities.

Value

data.frame with columns:

- datetime
- locality1_sensor1
- ...
- ...
- localityn_sensorn

Examples

```
example_tms_wideformat <- mc_reshape_wide(mc_data_example_raw, c("A6W79", "A2E32"),
                                          c("TMS_T1", "TMS_T2"))
```

`mc_save` *Save myClim object*

Description

This function was designed for saving the myClim data object to an .rds file, which can be later correctly loaded by any further version of myClim package with mc_load. This is the safest way how to store and share your myClim data.

Usage

```
mc_save(data, file)
```

Arguments

`data`	myClim object see myClim-package
`file`	path to output .rds file

Value

RDS file saved at the output path destination

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir)
mc_save(mc_data_example_agg, tmp_file)
file.remove(tmp_file)
```

mc_Sensor-class *Class for sensor definition*

Description

Sensor definitions are stored in mc_data_sensors.

Slots

`sensor_id` unique identifier of sensor (TMS_T1, TMS_T2, TMS_T3, TMS_TMSmoisture, ...)

`logger` name of logger (TMS, ThermoDatalogger, ...)

`physical` unit of sensor (T_C, TMSmoisture, moisture, RH_perc) (default NA)

`description` character info

`value_type` type of values (real, integer, logical) (default real)

`min_value` minimal value (default NA)

`max_value` maximal value (default NA)

`plot_color` color in pot (default "")

`plot_line_width` width of line in plot (default 1)

See Also

mc_data_sensors

mc_SensorMetadata-class *Class for sensor metadata*

Description

Class for sensor metadata

Details

`sensor_id` must be one of the defined id in myClim. see mc_data_sensors. It is useful to select on of predefined, because it makes plotting and calculaton easier. Through `sensor_id` myClim assign pre-deined physicyl units or plotting colors see mc_Sensor.

Slots

`sensor_id` unique identifier of sensor (TMS_T1, TMS_T2, TMS_T3, TMS_TMSmoisture, ...) mc_data_sensors e.g. TMS_T1, TMS_TMSmoisture, snow_fresh...

`name` character, could be same as `sensor_id` but also defined by function or user.

`height` character

`calibrated` logical - detect if sensor is calibrated

See Also

myClim-package, mc_LoggerMetadata, mc_data_sensors

`mc_TOMSTDataFormat-class`

Class for reading Tomst logger files

Description

Provides the key for the column in source files. Where is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

See Also

mc_DataFormat, mc_data_formats, mc_TOMSTJoinDataFormat

`mc_TOMSTJoinDataFormat-class`

Class for reading TMS join files

Description

Provides the key for the column in source files. Where is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

Details

TMS join file format is the output of IBOT internal post-processing of Tomst logger files.

See Also

mc_DataFormat,mc_data_formats,mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat

`myClimList` *Custom list for myClim object*

Description

Top level list for store myClim data. (see myClim-package) Rather service function used for checking, whether object is myClimList. The same time can be used to create standard R list from myClimList.

Usage

```
myClimList(metadata = NULL, localities = list())
```

Arguments

`metadata`	of data object
`localities`	list of licalities

Value

the list containing myClim object's metadata and localities

Index